ECHTZEITGEEIGNETE IMPLEMENTATION OBJEKTORIENTIERTER SOFTWARESYSTEME

Dr.-Ing. Egmont Woitzel*
Dipl.-Ing. Malte Köller
Universität Rostock
Fachbereich Elektrotechnik

ABSTRACT

Today object oriented methods are widely accepted as favorite tools for analysing, designing and implementing software systems. In special the problem of reusability of software could be solved by consequent application of object oriented techniques.
Nevertheless some serious drawbacks prevent the overall usage of object oriented programming in the field of embedded systems and real-time programming. First of all object oriented software systems require at run time much more memory and processing time than other programs, consumed by algorithms and data structures which implement late binding and dynamical object organisation, both essential for object oriented programming.
Due to the used implementation techniques the processing time requirements and response time of such systems often become non-predictable, so that in many cases real-time behavior could not be guaranteed.
This paper introduces a prototype of an interactive object oriented programming environment using a run-time system which guarantees determined response time. An additional advantage of that system is the optional usage of components which implement some time-critical organisation tasks at hardware level.

PROJEKTZIELE

In diesem Beitrag werden Ergebnisse des an der Universität Rostock, Fachbereich Elektrotechnik in Zusammenarbeit mit dem Lehr- und Forschungsgebiet Prozeßdatenverarbeitung (pdv) der RWTH Aachen ausgeführten und durch die Deutsche Forschungsgemeinschaft finanzierten Projekts "Echtzeitgeeignete Implementation objektorientierter Sprachsysteme" vorgestellt.
Im Mittelpunkt dieses Projekts standen Untersuchungen zu Datenstrukturen und Verfahren innerhalb von Laufzeitsystemen objektorientierter Sprachen, die die Potenzen der objektorientierten Softwareentwicklung, insbesondere der auf Vererbung und Polymorphie beruhenden Wiederverwendbarkeit von Softwaremoduln, auch im Umfeld von Echtzeitanwendungen anwendbar gestalten sollten. Entsprechend den allgemeinen Kriterien für industriell einsetzbare Echtzeitsoftware wurden folgende Forderungen aufgestellt:

- die zu entwickelnden Verfahren sollten ein deterministisches Zeitverhalten garantieren
- die zu entwicklenden Verfahren sollten universellen Charakter tragen, um in einer große Klasse objektorientierter Systeme anwendbar zu sein
- die Systemleistung sollte durch den optionalen Einsatz von Hardwarebaugruppen skaliert werden können
- der Applikationsprogrammierer sollte durch eine interaktive Umgebung bei Test und Inbetriebnahme unterstützt werden

DESIGNENTSCHEIDUNGEN

Um die Testung verschiedener Verfahren innerhalb eines objektorientierten Laufzeitsystems einfach zu ermöglichen, wurde als experimentelle Basis für die Untersuchungen ein interaktives objektorientiertes Programmiersystem implementiert. Der Entwurf dieses Systems wurde wesentlich von der oben genannten Forderung nach Universalität geprägt. Um die Verfahren auf eine möglicht große Klasse objektorientierter Systeme anwenden und gleichzeitig abschätzen zu können, in welchem Maße einzelne Merkmale eines objektorientierten Systems auf die Laufzeiteigenschaften eines Anwendungssystems rückwirken, sollte das Experimentalsystem relativ hohe Anforderungen an das Laufzeitsystem stellen. Im einzelnen sollte das zu implementierende System folgende Eigenschaften besitzen:

- analog zu SmallTalk sollten Zeiger auf Objekte nicht typisiert werden, so daß jede Methode an jedes Objekt gesendet werden darf; Methoden sind systemglobal bekannt
- die dynamische Erzeugung von Objekten sollte interaktiv möglich sein und als Methode implementiert werden, dadurch gelangt man zu einem Metaklassenansatz
- unabhängig von der noch nicht abgeschlossenen Diskussion über ihre Notwendigkeit sollte Mehrfachvererbung unterstützt werden, wobei auch das mehrfache Erben derselben Klasse möglich sein sollte
- Berücksichtigung interaktiv erweiterbarer Systeme, welche die Verwendung inkrementeller Compilationsverfahren fordern
- automatische Freigabe nicht mehr benötigter dynamischer Objekte zur Entlastung des Programmierers und Erhöhung der Robustheit
- Verschachtelung bzw. Einbettung von Objekten zur Vermeidung unnötiger Indirektionen und zur Minimierung der Anzahl der zu verwaltenden Speicherabschnitte
- explizite Unterscheidung statischer und dynamischer Objekte innerhalb eines Programms aus denselben Gründen
- der Programmierer sollte die Möglichkeit haben, spezielle Probleme auch außerhalb der objektorientierten Methodik zu lösen, dies führt zu einem hybriden Sprachansatz

Als Basis für die Implementation des Experimentalsystems wurde das Programmiersystem Forth gewählt, welches wegen seiner freien Erweiterbarkeit und der direkten Unterstützung einer interaktiven Arbeitsweise und inkrementellen Compilationstechnik beste Voraussetzungen für die Realisierung oben genannter Merkmale bot. Die Arbeiten wurden mit dem unter MS-DOS® arbeitenden System comFORTH 3.0 der FORTecH Software GmbH begonnen, die gegenwärtige Implementation arbeitet unter der jetzt verfügbaren MS-Windows™-Variante desselben Systems.

SYSTEMARCHITEKTUR

Schichtenaufbau

Auf Basis einer Analyse der für die Realisierung der oben genannten Eigenschaften des Sprachsystems notwendigen elementaren Funktionen des Laufzeitsystems wurden Kandidaten für eine optionale Hardwarerealisierung isoliert. Als besonders aussichtsreich erwiesen sich dabei die beiden Funktionsbereiche der Methodenbindung und der dynamischen Speicherverwaltung, die beide wesentlich für das problematische Laufzeitverhalten objektorientierter Systeme verantwortlich sind. Die Algorithmen und Datenstrukturen, die für die Bestimmung des einem konkreten Methodenaufruf zuzuordnendem Programmcodes sowie seiner Parametrisierung notwendig sind, wurden in einer hypothetischen Hardwareeinheit mit dem Namen Method Object Binder (MOB), diejenigen Bestandteile, die zur Verwaltung dynamischer Objekte notwendig sind, einer ebenfalls hypothetischen Hardwarebaugruppe mit dem Namen Object Oriented Memory Management Unit (ooMMU) zugeordnet.

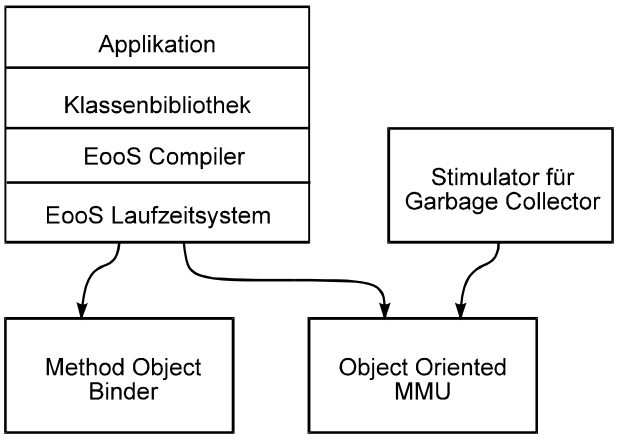

Bild 1: Schichtenaufbau des Experimentalsystems

Um die Testung verschiedener Verfahren zu vereinfachen und gleichzeitig die Einhaltung streng definierter Schnittstellen zu überwachen, wurde in der realisierten Lösung die Funktionen hypothetischer Hardwarebaugruppen bzw. Coprozessoren getrennt vom eigentlichen Laufzeitsystem implementiert. Unter MS-DOS wird die jeweilige Funktionalität als TSR-Programm, unter MS-Windows als dynamische Bibliothek (DLL) bereitgestellt.

Die einer echten Hardwarelösung inheränte Parallelität wird im Fall der ooMMU durch eine nebenläufige Implementation der Algorithmen simuliert. Unter MS-DOS kommt dazu ein spezielles Zeitscheibenverfahren zur Anwendung. Unter MS-Windows wird das im Betriebssystem implementierte kooperative Multitasking genutzt, um durch einen eigenständigen Prozeß eine ständige Stimulation der ooMMU vorzunehmen. An dieser Stelle soll betont werden, daß beide implementierten Lösungen harten Echtzeitforderungen nicht genügen. Allerdings erlauben sie eine Bestimmung der Aufrufhäufigkeiten der verschiedenen

Teilfunktionen, die als Basis für eine Abschätzung des Laufzeitverhaltens bei einer tatsächlichen Hardwareimplementation dienen können.
Die übrigen Funktionen des Experimentalsystems, speziell des Laufzeitsystems und des Compilers sind gemeinsam mit einer einfachen Klassenbibliothek und dem eigentlichen Applikationsprogramm in einer Erweiterung eines interaktiven Forth-Systems organisiert.

Funktion des Method Object Binder (MOB)

Der MOB verwaltet alle wesentlichen strukturellen Informationen über den Aufbau der innerhalb des Sprachsystems verfügbaren Klassen sowie der zu einem Zeitpunkt laufenden Methodenaufrufe. Im einzelnen nimmt der MOB folgende Teilfunktionen wahr:

- Speicherung aller für die Methodenbindung notwendigen Informationen zu einer Klasse (virtuelle Methodentabelle, Strukturtabelle)
- Bestimmung der Codeeintrittspunkte für einen Methodenaufruf über einem bestimmten Objekt
- Bestimmung der einer u. U. ererbten Methode zugänglichen Teilmenge der Instanzvariablen (des korrospondierden Teilobjekts)
- Bestimmung der Position einer Instanzvariablen innerhalb eines Objekts

Um ein deterministisches Zeitverhalten garantieren zu können, wurden alle Tabellen einstufig ausgelegt. Dies bedeutet beispielsweise für die virtuelle Methodentabelle einer Klasse, daß dort alle Informationen zu allen anwendbaren Methoden inklusive der ererbten und nicht redefinierten verfügbar sind. Dadurch entsteht zwar z. T. erhebliche Redundanz in den Verwaltungsstrukturen, die Zeit für die Ermitlung des zugehörigen Codes bleibt jedoch unabhängig von Form und Tiefe des Vererbungsbaumes.
Besondere Probleme verursacht hier die Fähigkeit zur mehrfachen Vererbung, die zu Schwierigkeiten bei der Vergabe systemweit eindeutiger Indizes für Methoden sowie bei der Bestimmung der Position der zugehörigen Instanzvariablen führt. Das Problem der Indexvergabe für Methoden ist insofern relevant, als dadurch die Gesamtgröße der virtuellen Methodentabellen wesentlich beeinflußt wird. Zur Minimierung des Speicherplatzbedarfs ist daher eine mehrfache Benutzung der selben Indizes durch unterschiedliche Methoden, die in keiner Klasse gemeinsam benutzt werden, vorzusehen. Durch die angestrebte inkrementelle Arbeitsweise des Compilers ist jedoch nicht auszuschließen, daß bei späteren Klassendefinitionen durch Ererben mehrerer Klassen aus bisher disjunkten Teilhierarchien Konflikte entstehen, die nur durch eine Neuvergabe der Methodenindizes gelöst werden können.
Zur eindeutigen Lokalisierung und Klassenbindung der in ein Objekt eingebetteten Instanzvariablen wurde eine sogenannte Strukturtabelle eingeführt, die nicht nur eine eindeutige Bestimmung von Teilobjekten in einem Objekt, sondern auch umgekehrt die Bestimmung desjenigen Objekts erlaubt, das Eigentümer eines gegebenen Teilobjekts ist.

Funktion der Object Oriented Memory Management Unit (ooMMU)

Durch die ooMMU werden alle dynamisch im System erzeugten Objekte verwaltet. Im Gegensatz zum MOB ist der ooMMU die innere Struktur zusammengesetzter Objekte jedoch nicht zugänglich. Die einzige relevante Information über den Aufbau eines Objekts stellt die Kenntnis derjenigen Zellen in einem Objekt dar, in denen Zeiger auf andere Objekte gespeichert werden. Diese Information wird der ooMMU als sogenannte Pointertabelle zur Verfügung gestellt.
Auf Basis dieser Informationen ist der ooMMU die kontinuierliche Durchmusterung des Objektsspeicherung nach nicht mehr referenzierten und damit nicht mehr benötigten Objekten möglich. Ausgangspunkt für die Suche nach Müll stellen dabei diejenigen Objekte dar, die aus statischen Objekten heraus referenziert werden. Derartige Referenzen werden durch das System bei der ooMMU an- bzw. abgemeldet und nach einem Reference-Count-Verfahren mitgezählt. Innerhalb des dynamischen Speichers wird kein derartiges Verfahren benutzt, da so zyklische Strukturen, die nicht mehr benötigt werden, nur schwer erkannt werden können. Die Bestimung der löschbaren Objekte erfolgt mit einem Mark&Sweep-Verfahren, das auch in derartigen Situationen fehlerfrei arbeitet.
Ein schwerwiegendes Problem stellt die bei längerer Nutzung des Speichers unweigerlich eintretende Fragmentierung des freien Speichers dar. Diesem Effekt wird durch einen periodischen Umkopierprozeß entegegengewirkt, der jeweils die nach einem Löschvorgang erhalten gebliebenen Objekte kompaktiert.
An dieser Stelle setzt die mögliche Hardwareimplementation an, die gerade diesen Umkopierprozeß durch Nutzung mehrerer mit einer DMA-Einheit gekoppelten Adreßrechenwerke auch unter Einhaltung harter Echtzeitbedingungen für die Software völlig transparent gestalten könnte.

SPRACHANSATZ

Der oberhalb des Laufzeitsystems implementierte Compiler ist als Erweiterung des Forth-Compilers konzipiert worden und ergänzt diesen durch Ausdrucksmittel für die Definition und Nutzung von Klassen. Forth wird dabei als Implementationssprache für Methoden unverändert weiterbenutzt. Ebenso ist im Sinne einer hybriden Sprache der Aufruf von Methoden aus nicht objektorientiert implementierten Programmbestandteilen problemlos möglich.
Die Definition neuer Klassen erfolgt auf Basis eines expliziten Metaklassenkonzepts, d. h. das Schlüsselwort für die Definition einer neuen Klasse `CLASS:` wendet im wesentlichen die Methode `NEW:` auf die angegebene Metaklasse an. Klassen werden nach folgenden Schema definiert, wobei mit einem * versehene Syntaxelemente ggf. mehrfach benutzt werden dürfen.

```
metaclass_name CLASS: new_class_name    \ neue Klasse anlegen
classname INHERIT: inheritname*         \ erben, ggf. mehrfach
                                        \ Deklarationen
     atomname VAR: varname*             \ atomare Instanzvariable
   classname PART: partname*            \ eingebettetes Objekt
          METHOD: methodname*           \ neue Methode deklarieren
                                        \ Methodencode
REDEFINE: methodname*                   \ neue oder ererbte Methode
          ...source text... ;REDEFINE   \ redefinieren
```

```
;CLASS                                    \ Abschluß der Klasse
```

Neue Metaklassen können durch Erben von der vordefinierten Klasse `META` definiert werden. Die von SmallTalk her bekannten Klasssenmethoden und Klassenvariablen können als Methoden bzw. Instanzvariablen von Metaklassen implementiert werden.
Innerhalb von Methodencode ist durch die Nennung von Variablennamen eine Bezugnahme auf die Adresse einer Instanzvariable möglich. Die Bezugnahme auf das eigene Objekt ist durch das Schlüsselwort `SELF` möglich, die Adressierung von Teilobjekten, die durch Vererbung eingebunden werden, ist durch das Schlüsselwort `PART` möglich, das auf einen `inheritname` angewendet werden darf. Für den gezielten Aufruf von Methodencode einer Elternklasse, der in der eigenen Klasse redefiniert wurde, kann durch Anwendung von `SUPER` auf einen `inheritname` die Methodenbindung auf die Elternklasse umgelenkt werden. Die generisch arbeitenden Operationen `OWNER` und `BUILDER` transformieren einen Objektzeiger in einen Zeiger auf das umschließende Objekt bzw. in das zugehörige Klassenobjekt.
Mit Hilfe dieser relativ schmalen Erweiterung des Forth-Compilers sind alle oben aufgezählten Anforderungen an das Laufzeitsystem stimulierbar, so daß diese Sprachoberfläche für den Test der skizzierten Verfahren geeignet ist.

AUSBLICK

Gegenwärtig befindet sich das Projekt in der abschließenden Evaluierungsphase, wobei insbesondere genaue Laufzeitmessungen einen Vergleich mit den Eigenschaften anderer objektorientierter und nicht objektorientierter Sprachsysteme gestatten werden. Dies ist insbesondere notwendig, um genauere Aussagen über die Effizienzgewinne treffen zu können, die bei einer Hardwarerealisierung der Basiskomponenten zu erwarten sind.
In einer anschließenden Phase ist die Erweiterung der Funktionalität des MOB und der ooMMU für den Einsatz in Multitasksystemen geplant.